PayPal™

Mobile Payments Library Developer Guide and Reference – Blackberry OS Edition

Last updated: August 2011

Table of Contents

Preface

The PayPal Mobile Payments Library provides secure, extensible, and scalable PayPal payment functionality to the Blackberry platform.

Purpose

The PayPal Mobile Payments Library provides an easy way for you to integrate payments into your Blackberry applications. You can download the library from X.com and include it in your application. With the library, you need only a few lines of code to integrate the payments library into your application.

When a buyer makes a payment, the library controls the checkout experience – logging in, reviewing, and completing the payment. After buyers complete their payments, the library returns the buyers to your application.

Scope

This document describes how to integrate the PayPal Mobile Payments Library with your application. You must create and provide your build to PayPal so PayPal can review your application before it is approved to accept payments by means of the library. The approval process is described later in the document.

OS and Hardware Support

The PayPal Mobile Payments Library supports Blackberry OS version 4.6 and higher. The list of supported Blackberry devices can be found here.

Revision History

The following table lists revisions made to the *PayPal Mobile Payments Library Developer Guide and Reference – Blackberry OS Edition*.

Version	Date Published	Description
1.0	August 2011	First publication

Where to Go for More Information

- *Adaptive Payments Developer Guide*
- *Sandbox User Guide*
- *Merchant Setup and Administration Guide*
- PayPal X Developer Network (x.com)

1. PayPal Mobile Payments Library

This section provides details about the Mobile Payments Library API, and it provides instructions and examples for integrating the library with your Blackberry application.

Mobile Payments Library API Reference

The flow of the library is:

1. Your application initializes the PayPal instance with your unique PayPal application ID.
2. You retrieve the PayPal instance.
3. You retrieve the PayPal button from the instance using the `getCheckoutButton` method. You can place the button on the screen.
4. You create and fill the `PayPalPayment` object to store the payment information.
5. *(Optional)* Your application enables Dynamic Amount Calculation (see step 7) to recalculate the payment amount, tax, currency, and shipping values depending on the shipping address. The recalculation occurs when the address is first received from the server and any time it is changed.
6. When buyers select the **Pay with PayPal** button, the library takes them through the PayPal Checkout experience.
7. *(Optional)* If you enabled dynamic amount calculation (step 5 above):
    a. When a buyer chooses an address for the payment, the library returns a call back to your application with the address information.
    b. Your application recalculates the payment and other amounts, based on the address.
    c. The library continues the buyer with the checkout experience, which uses the updated payment amount, tax, currency, and shipping values. This might include returning to library should point a) presents a custom view for address information.
8. After buyers complete their payments, the library triggers the corresponding event in the object implementing the `PayPalPaymentDelegate` interface, returning control to you with the results of the payment.

Adding the Library Jar File and Importing Classes in Eclipse

1. Right-click on your project and select “Properties”.
2. Select “Configure Build Path”.
3. Select the “Libraries” tab.
4. Select the “Add External JARs…” button.
5. Choose the “PayPal_MPL.jar” file from your folder structure and click “OK”.

Also, import the appropriate classes into your application classes. The following classes must be imported:

```
import com.paypal.blackberry.ui.CheckoutButton;
import com.paypal.blackberry.PayPal;
import com.paypal.blackberry. components.PayPalReceiverDetails;

import com.paypal.blackberry.PayPalPayment;
      or
import com.paypal.blackberry.AdvancedPayPalPayment;
import com.paypal.blackberry.util.BigDecimal;
```

NOTE: Since the PayPal Mobile Payments Library uses restricted APIs such as networking, all applications implementing it will need to include RIM signing.

Required Methods in the Mobile Payments Library

initWithAppID Method

The `initWithAppID` method creates and returns the PayPal object. You must pass in the unique application ID (appId) that PayPal has provided and the context. You can choose whether to use the live or sandbox server, or use non-networked (Demo) mode (see below).

```
static public PayPal initWithAppID(String appId,
     PayPalEnvironmentType environmentType)
```

An example of initializing the Library with this method is:

```
try{
PayPal ppObj = PayPal.initWithAppID("App-80W284485P519543T",
     PayPal.ENV_SANDBOX);
}catch(PayPalBlackberryLibraryException e) {}
```

Parameter	Description
`appId:`	*(Required)* PayPal Application ID from X.com. This is different for each server; thus, the `appId` is different for Live and Sandbox. Any `appId` value can be used when testing on None since the library does not contact the server when set to this.
`environmentType:`	*(Required)* Sets the PayPal environment to Live, Sandbox, or None. Allowable values are: • `ENV_LIVE` – Use the PayPal production servers. • `ENV_SANDBOX` – Use the PayPal testing servers. • `ENV_NONE` – Do not use any PayPal servers. Operate in demonstration mode, instead. Demonstration mode lets you view various payment flows without requiring production or test accounts on PayPal servers. Network calls within the library are simulated by using demonstration data held within the library.

NOTE: The Mobile Payments Library binds specific devices to specific application IDs, for enhanced security. For each of your application IDs, you must use a different sandbox account for each of your devices. If you log in with a different account on a device after binding, you receive the following error: "This app is attached to another PayPal account. To remove it, the account holder must visit PayPal.com and select Mobile Applications from the profile."

To switch a device or simulator to use a different sandbox account, go to the PayPal Sandbox website on your computer, log in with the account that was used on the device, select Profile > Mobile Applications, and then unbind the device from the application ID.

getCheckoutButton Method

You must obtain the **Pay with PayPal** payment button from the Mobile Payments Library. Create a local instance of `CheckoutButton`class (a subclass of `ButtonField`), which you can place in your application, and retrieve the button from the PayPal instance. As precondition, library should be initialized before a call to this method can be made (e.g. initWithAppId method should be called) or an exception will be thrown.

```
public synchronized Field getCheckoutButton(String fieldClassName,
     FieldSize buttonSize, TextType textType)
```

Example code of adding the Payment button to your application:

```
     CheckoutButtoncheckoutButton = (CheckoutButton)
         payPal.getCheckoutButton(PayPal.BUTTON_152x33,
         CheckoutButton.TEXT_DONATE);
     add(checkoutButton);
```

Parameter	Description
`fieldSize:`	*(Required)* Size and appearance of the **Pay with PayPal** button. Allowable values are: • `PayPal.BUTTON_152x33` • `PayPal.BUTTON_194x37` • `PayPal.BUTTON_278x43` • `PayPal.BUTTON_294x45` For images of the different button types, see "Enumerated Values in the Mobile Payments Library" on page 24
`textType:`	*(Required)* The type of button to be used. The type will determine the text that is to be used on the button. This has no bearing on the payment and affects only the button itself. Allowable values are: • `CheckoutButton.`*`TEXT_PAY`* • `CheckoutButton.`*`TEXT_DONATE`*

Implement the Delegate

If you want to be informed about payment status and payment result and take custom actions (e.g. display custom views) you should implement the delegate classes. The Library communicates back with your application using Java event listeners and delegates. When adding a button to your application, you add a `FieldChangeListener` to your buttons which executes when the user selects the corresponding item. You can implement `PayPalPaymentDelegate` to be informed immediately upon the successful completion, failure, or cancellation of a payment. Should you choose not to implement a custom delegate, you can use the default implementation, which doesn't do anything, provided by PayPal library.

To respond to the button press, implement a `FieldListener` on the PayPal Button. In this code, you create and manipulate the appropriate objects and settings to create your payment to be processed.

There are two types of payment delegates – `PayPalPaymentDelegate` and `PreapprovalDelegate`. `PayPalPaymentDelegate` is used to handle the processing of simple and advanced payments, while `PreapprovalDelegate` is used with preapproval payments.

In the following example, the buyer checks out with a simple payment for a single recipient:

```
FieldChangeListener listener = new FieldChangeListener() {
  public void fieldChanged(Field field, int context) {
     PayPalPayment newPayment = new PayPalPayment();
     // Set the payment amount
     newPayment.setCurrency("USD");
     // Set the sub total
     newPayment.setSubtotal(new BigDecimal(100));
     // Set the payment tax amount
     InvoiceData invoiceData = new InvoiceData();
     invoiceData.setTotalTax(new BigDecimal(9.65));
     invoiceData.setTotalShipping(new BigDecimal(2.00));
     newPayment.setInvoiceData(invoiceData);
     try {
           PayPal.getInstance().checkoutSimplePayment(newPayment,
               new SimplePayPalPaymentDelegate(),
               new SimplePaymentAdjuster(), PaymentType.SERVICES,
               PaymentSubType.B2B);
     } catch (PayPalBlackberryLibraryException e) {
           Dialog.alert("Error: " + e.getMessage());
     }
   }
};
```

The call to the `checkoutSimplePayment` method transfers control to the library which the user interacts with to complete the purchase. The library executes the transaction on the PayPal server and, based upon the result of the transaction, transfers control back to your application and calls the corresponding callback event to indicate the results of the transaction.

In the following example, the class implements the `PayPalPaymentDelegate` interface and responds to a `paymentCancelled()` event initiated by the Library by popping up the applications main screen

```
public class ReviewOrderScreenPaymentDelegate implements
      PayPalPaymentDelegate {
   …
   // setup class details
   …
   public void paymentCancelled() throws
```

```
            PayPalBlackberryLibraryPayPalPaymentDelegateException {

            UiApplication.getUiApplication().invokeLater(new
               Runnable() {
               public void run() {
                  Dialog.alert("payment cancelled");
                  UiApplication.getUiApplication(). pushScreen(new
                     DemoAppScreen());
               }}
            );
      }}
```

The following are the callback events triggered in the `PayPalPaymentDelegate` interface:

Method	Description
`paymentCancelled`	The payment was cancelled by the user.
`paymentFailed(String errorId, String errorMessage)`	The payment failed for the reason specified by error ID & `message`.
`paymentIncompleted`	The payment was incomplete.
`paymentCompleted`	The payment was successfully completed.

The `PreapprovalDelegate` is used in conjuncture with preapproval payments. The following callback events are triggered in the `PreapprovalDelegate` interface:

Method	Description
`preapprovalCancelled`	The preapproval was cancelled by the user.
`paymentFailed(String errorId, String errorMessage)`	The preapproval failed for the reason specified by error ID `& message`.
`preapprovalCompleted(String preapprovalKey)`	The preapproval was successfully completed.

Dynamic Amount Calculation

The Mobile Payments Library allows you to dynamically modify the payment information any time a buyer selects a shipping address. For instance, you might want to recalculate the tax amount based on the buyer's location.

To enable this, use the optional method `setDynamicAmountCalculationEnabled()` (see "Optional Methods" below). You must provide the logic that creates the new payment values based on the buyer's address. The library includes a `PaymentAdjuster` class for this.

To use this feature, one of your classes must implement "`PaymentAdjuster`", as well as implement "`Serializable`". For simplicity, we recommend creating a new class that does this. This class must include the following methods:

```
public Amounts adjustAmount(Address address, String currency,
      BigDecimal amount, BigDecimal tax, BigDecimal shipping);
```

Parameter	Description
`address`	The buyer's address that should be used when calculating adjusted tax and shipping amounts.
`currency`	The currency of the payment.
`amount`	The current subtotal amount.
`tax`	The current tax amount.
`shipping`	The current shipping amount.

Your method must return a new `Amounts` object (see the "Custom Objects in the Mobile Payments Library" section). This object contains the new currency and amounts.

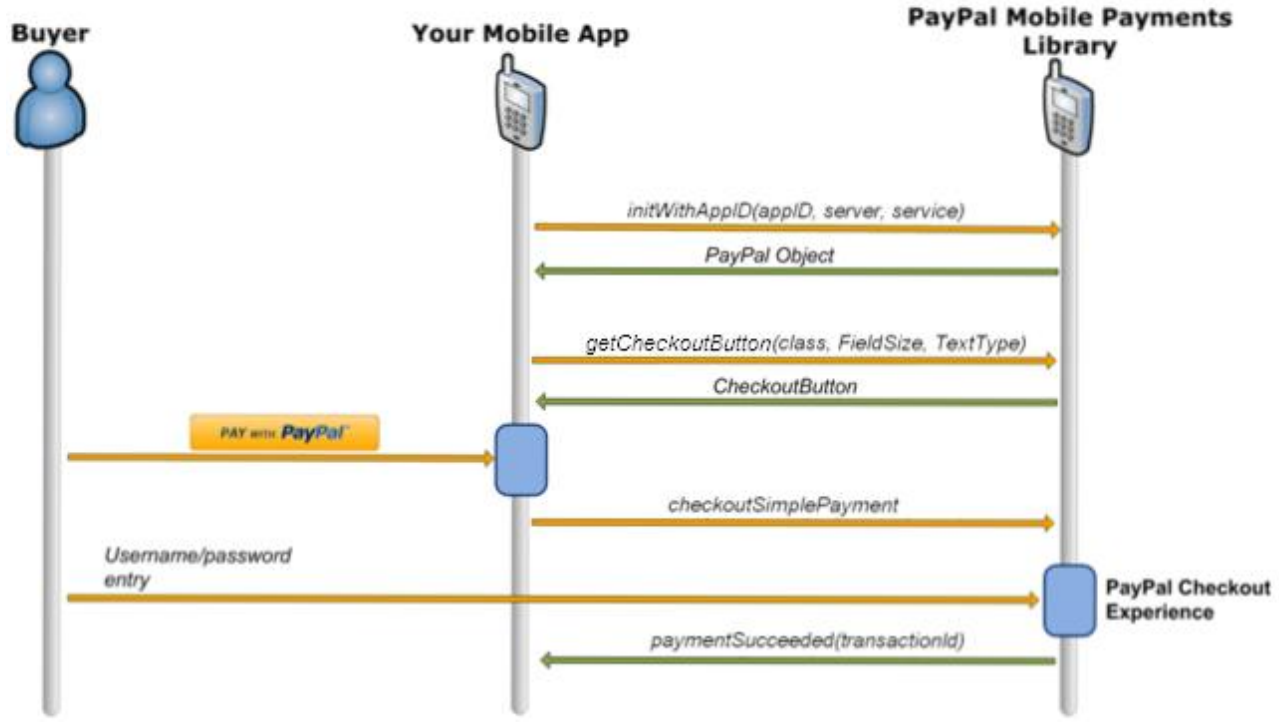
```
public Vector<PayPalReceiverDetails> adjustAmountsAdvanced(Address
     address, BigDecimal currency, Vector< PayPalReceiverDetails>
     receivers);
```

Parameter	Description
`address`	The buyer's address that should be used when calculating adjusted tax and shipping amounts.
`currency`	The currency of the payment.
`receivers`	A collection of current receivers and the amounts associated with each receiver.

Your method must return a new `Vector<PayPalReceiverDetails>` to update the library with adjusted amounts for each receiver. (See the "Custom Objects in the Mobile Payments Library" section).

Method Sequence

The following diagram illustrates the sequence of methods required to implement the checkout experience.


Buyer
Your Mobile App
PayPal Mobile Payments Library
initWithAppID(appID, server, service)
PayPal Object
getCheckoutButton(class, FieldSize, TextType)
CheckoutButton
PAY WITH PayPal
checkoutSimplePayment
Username/password entry
PayPal Checkout Experience
paymentSucceeded(transactionId)

Optional Methods in the Mobile Payments Library

getInstance Method

This method returns the singleton PayPal object.

```
PayPal payPal = PayPal.getInstance();
```

setLang Method

```
payPal.setLang (String localeCode);
```

Enable / Disable Shipping Method

This method lets buyers include display of shipping addresses in the library. With shipping enabled, buyers can choose an address from the list available in their PayPal accounts. The chosen shipping address is then used for the payment. Shipping is enabled by default.

```
payPal.setShippingEnabled(boolean isEnabled);
```

setFeesPayer Method

This method is valid only for Personal payments. Call this method to set who pays any fees, by default. If you do not call this method, the receiver pays any fees by default.

```
payPal.setFeePaidByReceiver (boolean feePaidByReceiver);
```

setDynamicAmountCalculationEnabled Method

This method lets you recalculate the payment amount, tax, currency, and shipping values based on the shipping address chosen by a buyer. If you use this method to enable dynamic amount calculation before the checkout starts, the library will dynamically update the payment based on logic you provide (see above).

NOTE: If shipping is not enabled, this method is ignored.

```
payPal.setDynamicAmountCalculationEnabled(boolean enabled);
```

After the Payment

After the payment is completed, the Mobile Payments Library returns the payKey. A number of other features are also available to you to enable you to further deal with the payment: Instant Payment Notification, Transaction Details, and Refunds.

Instant Payment Notification

Instant Payment Notification (IPN) is PayPal's message service that sends a notification when a transaction is affected. You can integrate IPN with your systems to automate and manage your back office. More details and documentation are available at: www.paypal.com/ipn

Transaction Details

You can integrate with the PayPal `PaymentDetails` API to retrieve details on a payment based on the payKey. More details and documentation are available at: https://cms.paypal.com/cms_content/US/en_US/files/developer/PP_AdaptivePayments.pdf

Refunds

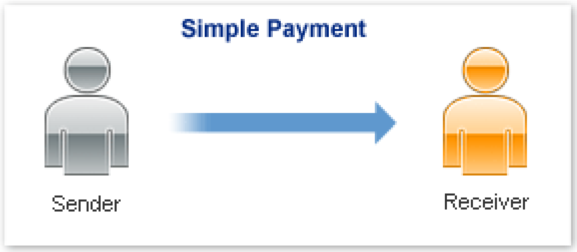
Refunds can be supported by manual refund through the PayPal account interface or by means of the Refund API. More details and documentation are available at: https://cms.paypal.com/cms_content/US/en_US/files/developer/PP_AdaptivePayments.pdf

Simple, Parallel, and Chained Payments

Simple payments have a single recipient. Parallel and chained payments have multiple recipients and differ in the how the payments are split.

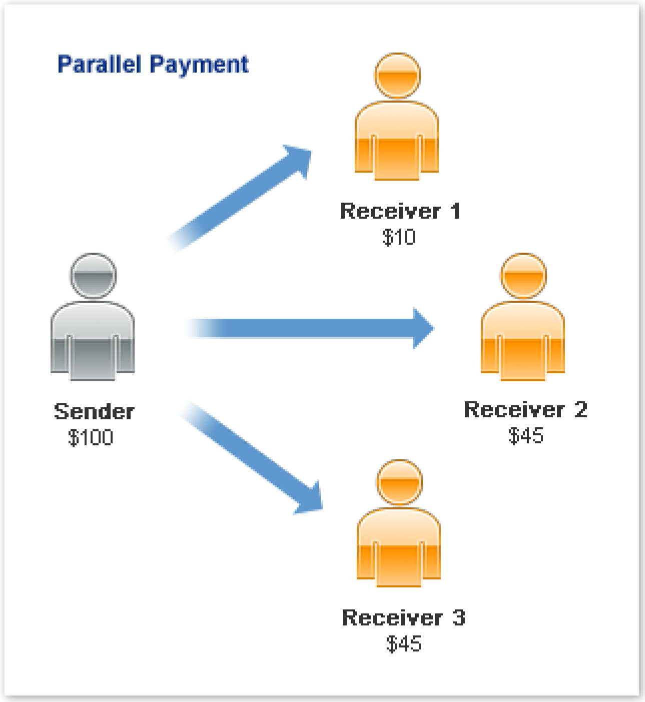
Simple Payments

Simple payments use the `PayPalPayment` object, which supports a payment to only a single recipient.



Parallel Payments

Parallel Payments allow you to make payments for any amount to any number of recipients. A parallel payment is created by making a payment to multiple recipients with no primary recipient. From the end-user's standpoint, a parallel payment will affect the UI by showing the details for each recipient. Contrary to chained payments, the recipients of a parallel payment are not linked together in terms of amount.

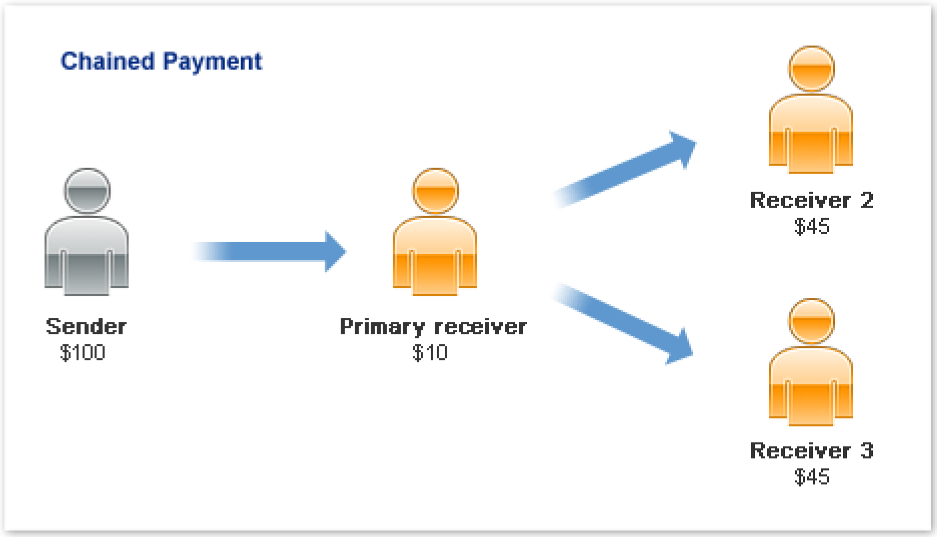


Chained Payments

A chained payment is a payment from a sender that is indirectly parallel among multiple receivers. It is an extension of a typical payment from a sender to a receiver; however, a receiver, known as the *primary receiver*, passes part of the payment to other receivers, who are called *secondary receivers*.

NOTE: Chained payments require a specific permission level on the part of the API caller and merchant. For information, refer to the section "Adaptive Payments Permission Levels" in the *Adaptive Payments Developer Guide*.

You can have at most one primary receiver and from 1 to 5 secondary receivers. Chained payments are useful in cases when the primary receiver acts as an agent for other receivers. The sender deals only with the primary receiver and does not know about the secondary receivers, including how a payment is parallel among receivers. The following example shows a sender making a payment of $100:



In this example, the primary receiver receives $100 from the sender’s perspective; however, the primary receiver actually receives only $10 and passes a total of $90 to secondary receivers (Receiver 2 and Receiver 3).

NOTE: The scenario above is an example only and does not take PayPal fees into account.

Preapprovals

The PayPal Mobile Payments Library lets you set up and obtain authorization in advance from buyers for future payments to you without requiring buyers to authorize each payment individually. For example, you might use the library to establish preapproval agreements for subscriptions to mobile content, such as mobile streaming audio or video. Or, you might use the library to establish preapproval agreements for payments to gain access to higher levels of difficulty in mobile games.

How Preapprovals Work

There are 3 steps to setting up and using preapprovals.

1. Obtain a *pending preapproval key* from PayPal.

   From your application, create a `PreapprovalPayment` object and a `PreapprovalRequest` object with the terms of your preapproval agreement.

2. Obtain authorization from the buyer for the preapproval agreement.

   Obtain the preapproval key and set it with the `setPreapprovalKey` method. After this is set, invoke the Preapproval flow by calling the method `preapprove` method of the PayPal instance, passing it objects that implement the `PreapprovalPayment` and `PreapprovalRequest`. The library launches the preapproval checkout experience and, if it is processed successfully, the `preapprovalSucceeded` method is called. If the processing fails, the `preapprovalFailed` method is invoked.

3. Take payments from the buyer under the terms of the preapproval agreement.

   From your web server, send a `Pay` request to PayPal with the buyer's confirmed preapproval key.

For more information about the `Preapproval` and `Pay` requests, see the *Adaptive Payments Developer Guide*.

About Preapproval Keys

Preapproval keys uniquely identify your preapproval agreements. Preapproval keys that you obtain by using the `Preapproval` API identify your pending preapproval agreements. No buyers have yet agreed to them. Pending approval keys remain valid for 3 hours before expiring without confirmation from buyers.

To launch the preapproval checkout experience to confirm a buyer's agreement to a pending preapproval, set the preapproval key in the PayPal instance using the `setPreapprovalKey` method. This preapproval key is retuned if the buyer completes the preapproval checkout. Maintain a record of buyers and their confirmed preapproval keys on your web server. Later on your web server, take payments from buyers by sending `Pay` requests with buyers' preapproval keys to PayPal.

About Preapproval Pins

Confirmed preapproval keys let you take payments from buyers without requiring them to log in to PayPal to authorize payments individually. Depending on your business model, you may want to obtain consent quickly from buyers before you take individual payments. Preapproval PINs are special codes that buyers enter to authorize preapproved payments individually without logging in to PayPal.

For example, you might have a mobile game that requires payment from buyers to enter a higher level of difficulty. You could take the payment, without notice, when the buyer enters the higher level. However, the buyer might dispute the payment later, despite the preapproval agreement and the automatic payment notice from PayPal. Obtain a buyer's consent before you take the entrance fee to help improve the buying experience.

Specify that you want your preapprovals to use preapproval PINs when you send `Preapproval` requests from your web server to PayPal. Set the `PreapprovalRequest.pinType` to `REQUIRED`. PayPal returns preapproval keys that require buyers to create preapproval PINs during preapproval checkout.

Later, when you take payments by using a buyer's confirmed preapproval key, prompt the buyer for the preapproval PIN. Pass the buyer's PIN to PayPal when you send the `Pay` request from your web server. PayPal recommends that you display the payment reason and payment amount when you prompt buyers for their preapproval PINs.

Sample Call

After you obtain a pending approval key, construct a `PreapprovalPayment` object that includes the key and the merchant's name. Then, use the `checkoutPreapproval()` method to execute the call.

```
PreapprovalPayment preapproval = new PayPalPreapproval();
preapproval.setCurrency("USD"); preapproval.setMerchantName("Preapproval
Merchant");

try {
     PayPal.getInstance().checkoutPreapproval(preapproval, new
          PreapprovalDelegate(), new PaymentAdjuster(),
          PaymentType.SERVICES, PaymentSubType.B2B);
} catch (PayPalBlackberryLibraryException e) {
     Dialog.alert("Error: " + e.getMessage());
}
```

Custom Objects in the Mobile Payments Library

The Mobile Payments Library includes custom objects for passing information between the library and your application during checkout. Use these objects if you enable dynamic amount calculation by calling the `DynamicAmountUpdate` method.

MEPAddress

This object is passed to your `PaymentAdjuster class` in the `adjustAmount` method. Use this address to update the payment amount, tax, currency, and shipping values of the payment. Then, the buyer continues to check out with the new amounts.

Method	Description
`String getStreet1()`	First line of the street address.
`String getStreet2()`	Second line of the street address.
`String getCity()`	Name of the city.
`String getState()`	Name of the state or province.
`String getPostalcode()`	U.S. ZIP code or other country-specific postal code.
`String getCountrycode()`	The 2-character country code.
`String getCountry()`	The name of the country.

MEPAmounts

This object is returned to the library by the `adjustAmount` method of your `PaymentAdjuster` class. This object contains the values for the updated payment.

Property	Description
`setCurrency(String currency)`	(*Optional*) Currency code of the amount. Defaults to USD.
`setPaymentAmount(BigDecimal paymentAmount)`	(*Required*) Amount of the payment before tax or shipping.
`setTax(BigDecimal tax)`	(*Optional*) Tax amount associated with the item.
`setShipping(BigDecimal shipping)`	*(Optional)* Shipping amount for the item

PayPalPayment

This object is passed to the library when the specific checkout method (`checkoutSimplePayment`, `checkoutPreapprovalPayment`, etc.) is called. This object contains all the values for a payment.

Method	Description
`setCurrencyType(String currencyType)`	*(Optional)* – Currency code for the payment. Defaults to USD if not set.
`setSubtotal(BigDecimal subtotal)`	*(Required)* – Subtotal for the payment.
`setRecipient (String recipient)`	*(Required)* – The email address or phone number of the payment's recipient.
`setMerchantName(String name)`	*(Optional)* – Displayed at the top of the library screen. If not set, displays the recipient email or phone number string instead.
`setInvoiceData(PayPalInvoiceData invoiceData)`	*(Optional)* – Holds information such as tax and shipping amounts as well as an optional breakdown of the items included.
`setCustomID(String customID)`	*(Optional)* – The merchant's custom ID. This can be used as a pass through field which will later be available on the IPN.
`setIpnUrl(String IpnUrl)`	*(Optional)* – The IPN URL.
`setDescription(String description)`	*(Optional)* – Description of the item. Defaults to "Item" if not set.

PayPalAdvancedPayment

This object is passed to the library when the associated non-simple Checkout method is called. This object contains all the values for a payment

Method	Description
`setCurrencyType(String currencyType)`	*(Optional)* – Currency code for the payment. Defaults to USD if not set.
`setReceivers(Vector PayPalReceiverDetails)`	*(Required)* – Your payment must have at minimum one receiver. The receiver itself should be set up before adding it to the payment. See "PayPalReceiverDetails" below for more details. The `getReceivers ()` call can be used to alter the current Vector.
`setIpnUrl(String ipnUrl)`	*(Optional)* – The IPN URL.
`setMemo(String memo)`	*(Optional)* – Note for the payment.

PayPalInvoiceData

This is an optional object to be set for any receiver. The `PayPalPayment` (simple payment) has only one receiver, and the `InvoiceData` is added directly to the payment object itself. The `PayPalAdvancedPayment` can support multiple receivers, each of which can have an `InvoiceData` added to it.

While `InvoiceData` is an optional parameter for any given receiver, once added the `InvoiceData` must be populated with certain required parameters (see below).

Method	Description
`setTax(BigDecimal tax)`	*(Required)* – Tax amount to be used for the payment. This can be updated after the checkout flow has been started if dynamic amount calculation is enabled. The amount can be 0.
`setShipping(BigDecimal shipping)`	*(Required)* – Shipping amount to be used for the payment. This can be updated after the checkout flow has been started if dynamic amount calculation is enabled. The amount can be 0.
`setInvoiceItems(Vector invoiceItems)`	*(Required)* – Sets the list of items in the invoice. See "PayPalInvoiceItem" below for more details. These items do not affect the total amount of the payment but must equal the subtotal.

PayPalInvoiceItem

These items can be added to any `PayPalInvoiceData`.

NOTE: The `Price` and `Quantity` must multiply together correctly to equal the `Price`. The `TotalPrice`s of all `invoiceItems` of a `PayPalPayment` or a `PayPalAdvancedPayment` must equal the subtotal of the payment.

Method	Description
`setName(String name)`	*(Optional)* – The name of the item.
`setID(String ID)`	*(Optional)* – A unique ID for the item.
`setTotalPrice(BigDecimal price)`	*(Required)* – The total cost of this item (unit cost * quantity).
`setQuantity(int quantity)`	*(Required)* – The quantity of this item.

PayPalReceiverDetails

This object is used in the `AdvancedPayPalPayment` to identify a single recipient.

Method	Description
`setReceivers(Vector PayPalReceiverDetails)`	*(Required)* – The email address or phone number of this recipient.
`setSubtotal(BigDecimal subtotal)`	*(Required)* – The subtotal of the payment to this recipient.
`setIsPrimary(boolean isPrimary)`	*(Optional)* – Indicates whether or not the receiver is the primary receiver of the payment. A payment with multiple recipients may specify the primary recipient of the payment to denote a chained payment. Payments with multiple recipients but no primary receivers are parallel payments. See the section "Chained, Parallel, and Simple Payments" for more details.
`setPaymentType(PaymentType paymentType)`	*(Optional)* – See enumeration values below.
`setPaymentSubType(PaymentSubType paymentSubType)`	*(Optional)* – See enumeration values below.
`setInvoiceData(PayPalInvoiceData invoiceData)`	*(Optional)* – The `InvoiceData` holds information such as tax and shipping amounts as well as an optional breakdown of the items included. See "PayPalInvoiceData" above for more details.
`setCustomID(String customID)`	*(Optional)* – The recipient's custom ID.
`setMerchantName(String merchantName)`	*(Optional)* – Displayed at the top of the library screen. If not set, displays the recipient email or phone number string instead.

Enumerated Values in the Mobile Payments Library

The enumerated values supported by various methods in the library are:

PayPalServerType

- `ENV_LIVE` – Use the PayPal production servers.
- `ENV_SANDBOX` – Use the PayPal testing servers.
- `ENV_NONE` – Do not use any PayPal servers. Operate in demonstration mode, instead. Demonstration mode lets you view various payment flows without requiring production or test accounts on PayPal servers. Network calls within the library are simulated by using demonstration data held within the library.

NOTE: `ENV_LIVE` does not support simulators.

FieldSize

`BUTTON_118x24`

`BUTTON_152x33`

`BUTTON_194x37`

`BUTTON_278x43`

`BUTTON_294x45`

NOTE: If the `TextType parameter` is set to "Donate", the word "Pay" in the above buttons is replaced by "Donate." The language of the button will also change based on the language you pass into the `setLang` method or the auto detected language on the phone.

PaymentType

```
GOOD
PERSONAL
SERVICE
```

NOTE: For `Personal` payment types, the PayPal Checkout experience differs slightly from other payment types. Additionally for `Personal` payment types, senders in some cases can choose who pays any fees – the sender or the recipient. In India and Germany, recipients always pay any fees.

For more information, see "setFeePaidByReceiver Method" on page 15.

PaymentSubType

```
AffiliatePayments
B2B
ChildCare
Donations
Entertainment
EventPlanning
GeneralContractors
Government
Insurance
Invoice
Medical
Mortgage
Payroll
Rebates
Refunds
Reimbursements
Remittances
Rent
Tourism
Transfer
Tuition
Utilities
```

NOTE: The payment subtype for a SERVICE type payment. Applicable only if you have been approved for special pricing plans. For any `PaymentType` other than SERVICE, or if you have not been approved for special pricing, subtype is not processed.

TextType

- `Pay` – Used for payments; button will have text "Pay" or "Pay with PayPal".
- `Donate` – Used for donations; button will have text "Donate" or "Donate with PayPal".

Localization Support in the Mobile Payments Library

To change the language in the library, call `setLang (String localeCode)` on the PayPal object. Pass through the desired locale code (for example, for US English, use `en_US`). If you do not use this method, the library defaults to the locale to which the device is set.

The library supports the following regions:

Language (Country or Region)	**Locale Code**
Chinese (Hong Kong)	`zh_HK`
Chinese (Taiwan)	`zh_TW`
Dutch (Belgium)	`nl_BE`
Dutch (Netherlands)	`nl_NL`
English (Australia)	`en_AU`
English (Belgium)	`en_BE`
English (Canada)	`en_CA`
English (France)	`en_FR`
English (Germany)	`en_DE`
English (Great Britain)	`en_GB`
English (Hong Kong)	`en_HK`
English (India)	`en_IN`
English (Japan)	`en_JP`
English (Mexico)	`en_MX`
English (Netherlands)	`en_NL`
English (Poland)	`en_PL`
English (Singapore)	`en_SG`

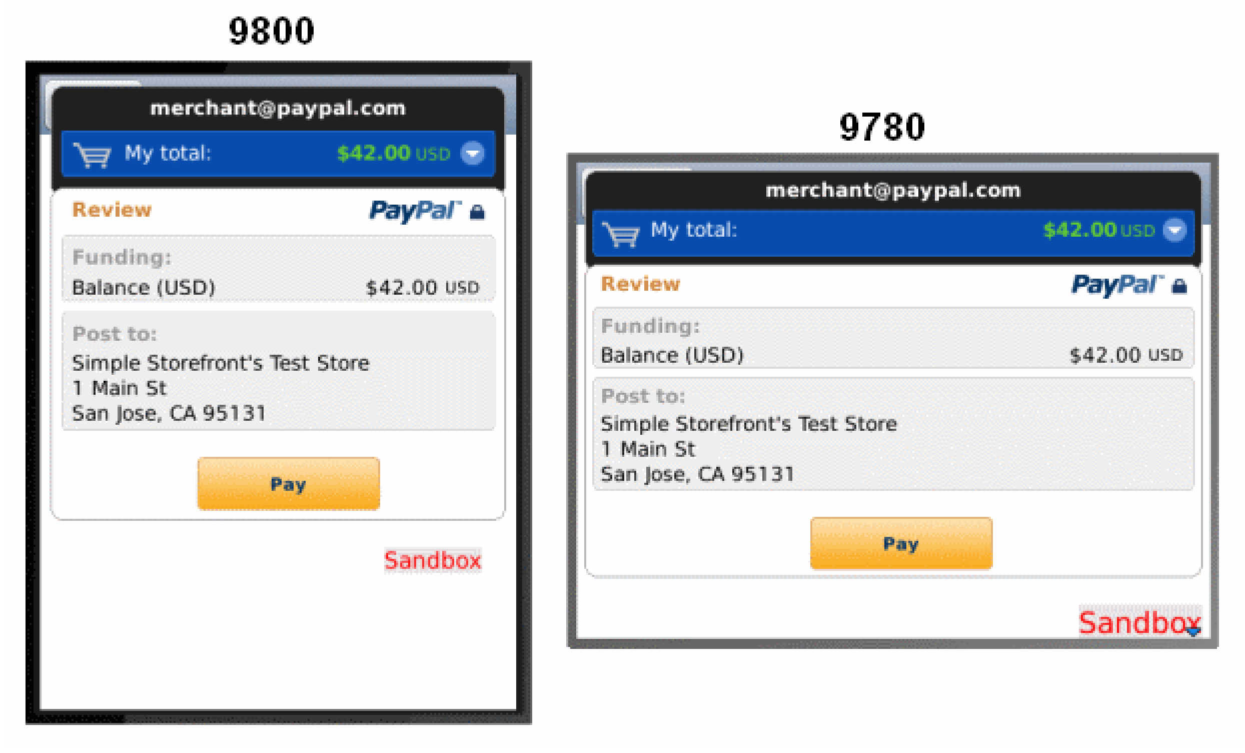
Language (Country or Region)	Locale Code
English (Spain)	`en_ES`
English (Switzerland)	`en_CH`
English (Taiwan)	`en_TW`
English (United States)	`en_US`
French (Belgium)	`fr_BE`
French (Canada)	`fr_CA`
French (France)	`fr_FR`
French (Switzerland)	`fr_CH`
German (Austria)	`de_AT`
German (Germany)	`de_DE`
German (Switzerland)	`de_CH`
Italian (Italy)	`it_IT`
Japanese (Japan)	`ja_JP`
Spanish (Argentina)	`es_AR`
Spanish (Mexico)	`es_MX`
Spanish (Spain)	`es_ES`
Polish (Poland)	`pl_PL`
Portuguese (Brazil)	`pt_BR`

2. The Checkout Experience with the Mobile Payments Library

The following screen shots illustrate several different PayPal Checkout experiences that occur after buyers click the PayPal button that your application obtains from the library by means of the `getCheckoutButton()` method.

Checkout Experience #1 – Goods or Services with Shipping

PaymentType = GOOD or SERVICE / Shipping = enabled



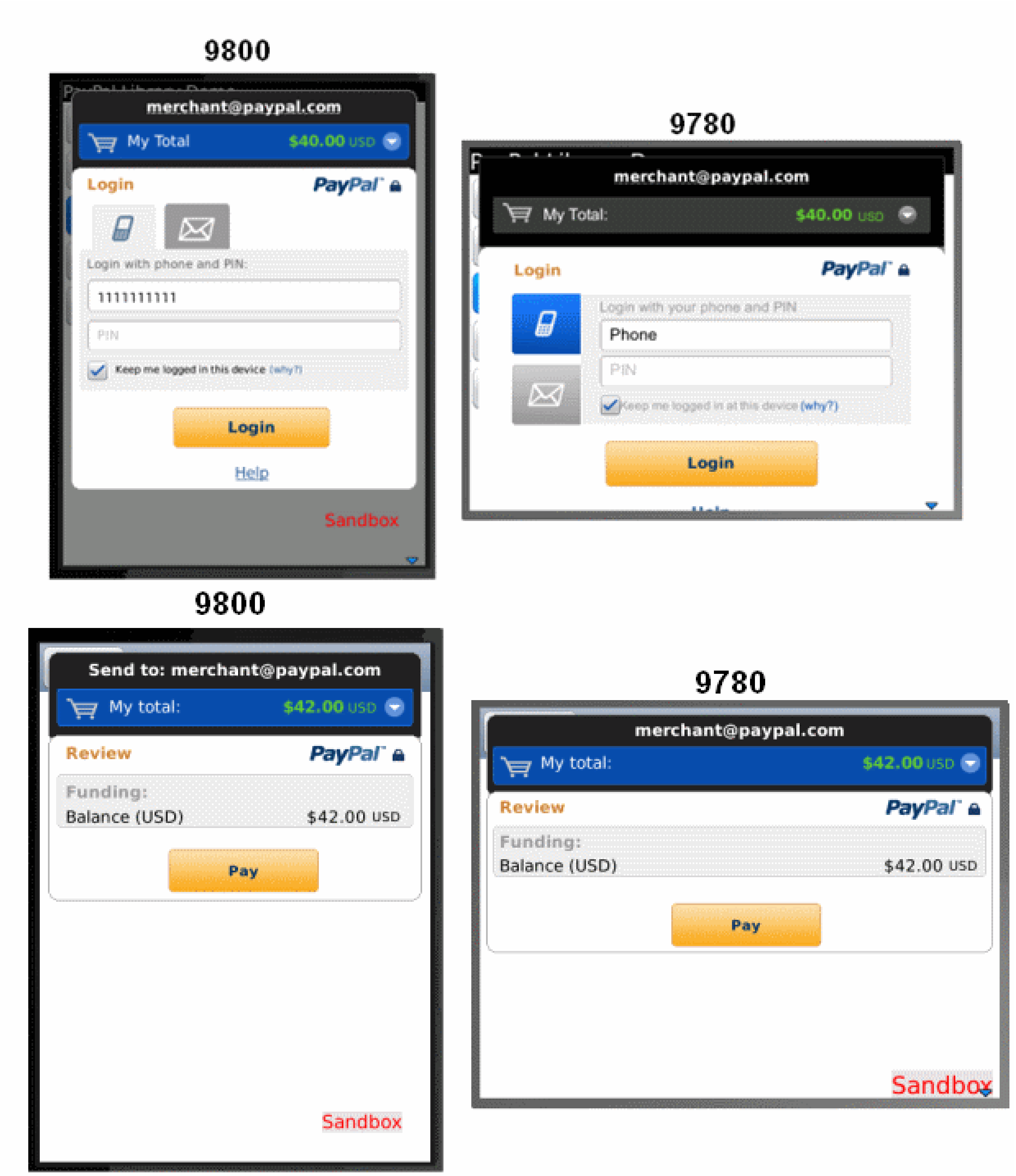
merchant@paypal.com
My total: $42.00 USD
Review PayPal
Funding:
Balance (USD) $42.00 USD
Post to:
Simple Storefront's Test Store
1 Main St
San Jose, CA 95131
Pay
Sandbox

merchant@paypal.com
My total: $42.00 USD
Review PayPal
Funding:
Balance (USD) $42.00 USD
Post to:
Simple Storefront's Test Store
1 Main St
San Jose, CA 95131
Pay
Sandbox

In the preceding experience, buyers enter their PayPal login credentials in the *Log In To PayPal* screen. Then, they can review details of the payment in the second screen and the change funding source or shipping address. If satisfied, buyers click **Pay** to complete the payment.

Checkout Experience #2 – Goods or Services without Shipping

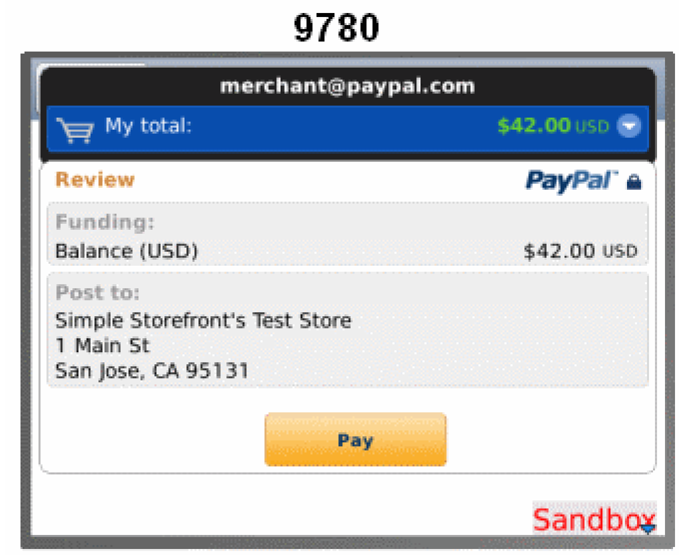
PaymentType = GOODS or SERVICE / Shipping = disabled



In the case, shipping is not required (that is, manual pick up of goods or services). Shipping is disabled by a call to the `setShippingEnabled()` library method. Buyers enter their PayPal login credentials and directly pay by clicking the **Pay** button on the first screen. Buyers can review funding choices by clicking the **Review** button on the same page.

Checkout Experience #3 – Donations

PaymentSubType = Donations / Shipping = enabled



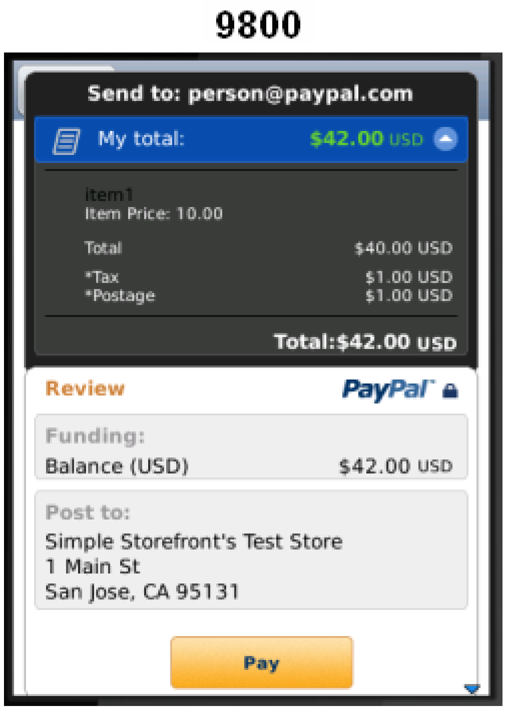
merchant@paypal.com

My total: $42.00 USD

Review PayPal

Funding:
Balance (USD) $42.00 USD

Post to:
Simple Storefront's Test Store
1 Main St
San Jose, CA 95131



Send to: person@paypal.com

My total: $42.00 USD

item1
Item Price: 10.00

Total $40.00 USD
*Tax $1.00 USD
*Postage $1.00 USD

Total:$42.00 USD

Review PayPal

Funding:
Balance (USD) $42.00 USD

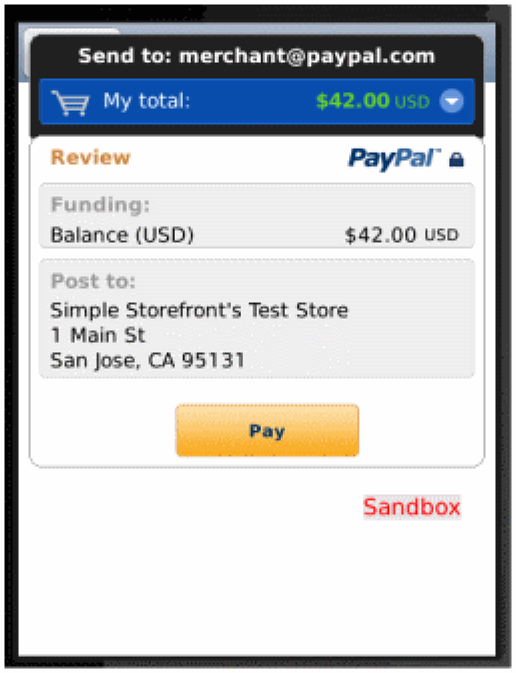
Post to:
Simple Storefront's Test Store
1 Main St
San Jose, CA 95131

In the preceding experience, buyers make a donation to a charity or other cause. In this context, the charity or cause wants to leverage PayPal members' addresses as mailing addresses for donation receipts. By enabling shipping in the library, buyers are presented with their primary mailing address but can choose another mailing address from the ones in their PayPal accounts.

Checkout Experience #4 – Personal Send Money Payments

Payment Type = PERSONAL / Shipping = disabled

9800


Send to: merchant@paypal.com

My total: $42.00 USD

Review

Funding:

Balance (USD) $42.00 USD

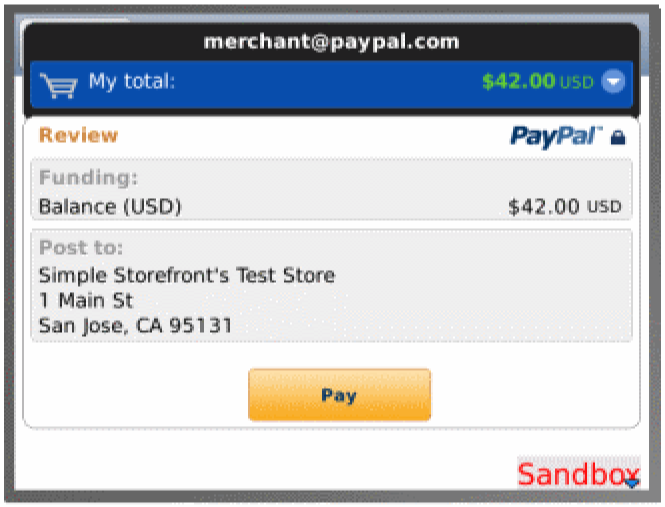
Post to:

Simple Storefront's Test Store
1 Main St
San Jose, CA 95131

Pay

Sandbox

9780


merchant@paypal.com

My total: $42.00 USD

Review

Funding:

Balance (USD) $42.00 USD

Post to:

Simple Storefront's Test Store
1 Main St
San Jose, CA 95131

Pay

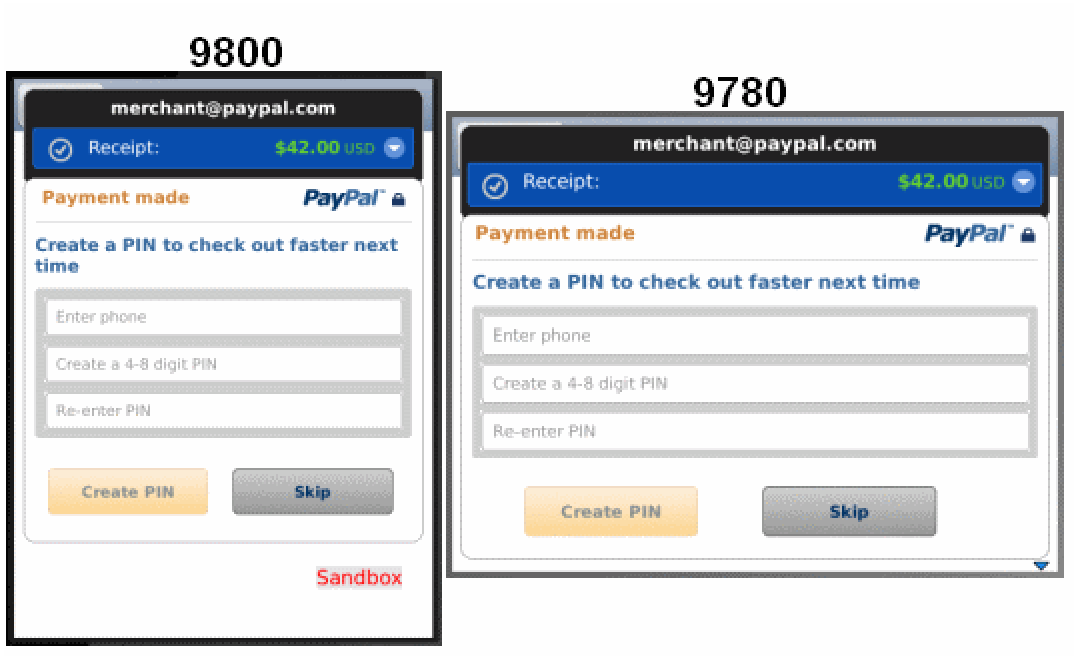
Sandbox

In the preceding experience, PayPal members make personal payments to other PayPal members. There are no transaction fees when it is funded by PayPal balance or by a bank account on file. The transaction carries a fee when it is funded by payment card – debit or credit or PayPal Credit cards. In some cases, senders choose who pays any fees – sender or recipient. In India and Germany, recipients always pay any fees.

For more information on PayPal Send Money and pricing, refer to:

https://cms.paypal.com/us/cgi-bin/?cmd=_render-content&content_ID=marketing_us/send_money

Checkout Experience #5 – Create Pin



In the preceding experience, a PayPal member has just completed a payment and does not currently have a PIN associated with his or her account. By following the on-screen instructions, the user can associate the account with a phone number and PIN for easier login in the future.

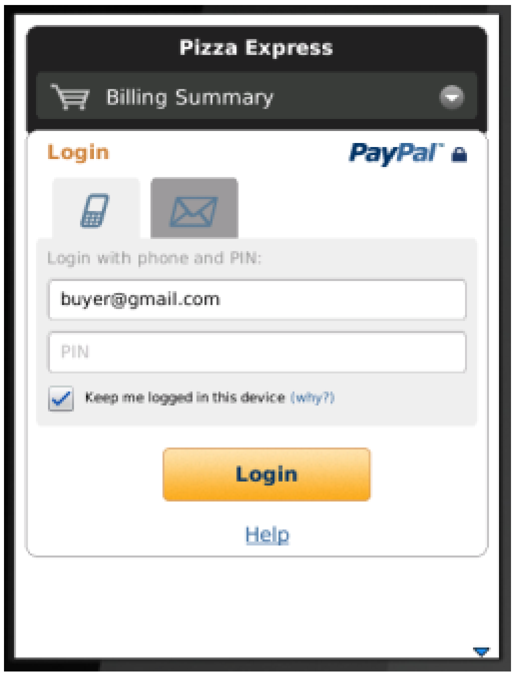
Upon successful creation of the PIN, the user is returned to your application, triggering the paymentSucceeded() delegate callback.

Checkout Experience #6 – Preapproval

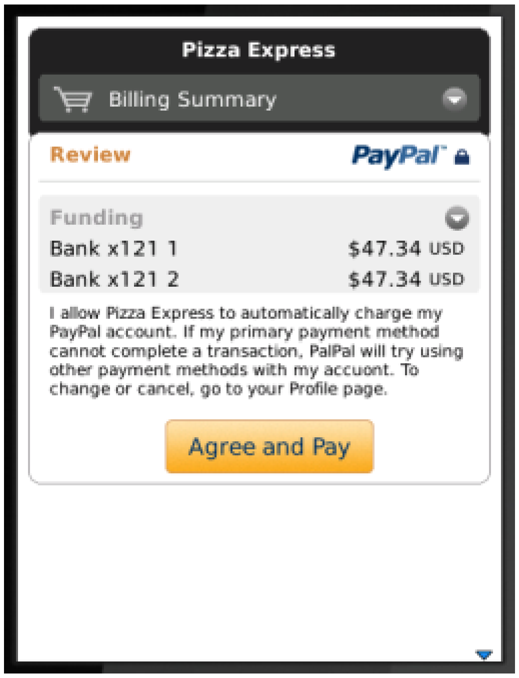
In this experience, you started the PayPal preapproval flow by calling the `checkoutPreapproval` method, as discussed under "Preapprovals" on page 17.

Basic Preapproval Checkout

Login Screen



Agree and Pay Screen



Bank x121 1	$47.34 USD
Bank x121 2	$47.34 USD

I allow Pizza Express to automatically charge my PayPal account. If my primary payment method cannot complete a transaction, PalPal will try using other payment methods with my account. To change or cancel, go to your Profile page.

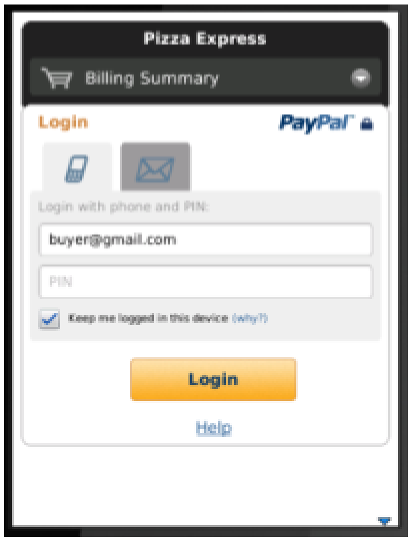
During a preapproval checkout, the buyer agrees to the terms of a preapproval agreement. The agreement authorizes you to take payments without requiring the buyer to log in to PayPal to authorize the payments individually. After the buyer completes the checkout, PayPal returns the buyer's *confirmed preapproval key* to your mobile application.

Use the buyer's *confirmed preapproval key* to take the preapproved payments. The library does not take the payments for you. After UI control returns to your mobile application, store the buyer's preapproval key on you web server. Then, take your first preapproved payment by sending a `Pay` request with the buyer's preapproval key from your web server to PayPal.

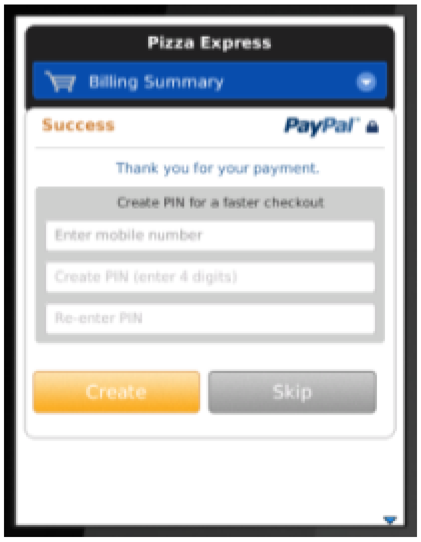
Creating Preapproval PINs During Preapproval Checkout

Depending on your business model, you may require buyers to create *preapproval PINs* during preapproval checkout. Preapproval PINs are special codes that buyers specify during checkout to let them consent quickly later to individual payments. If your preapproval agreements require PINs, PayPal displays the optional “Create a Code” screen during preapproval checkout.

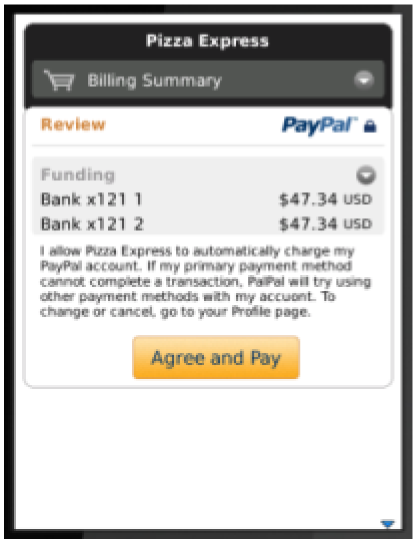
Login Screen



Create a Code Screen


Pizza Express
Billing Summary
Success
PayPal
Thank you for your payment.
Create PIN for a faster checkout
Enter mobile number
Create PIN (enter 4 digits)
Re-enter PIN
Create
Skip

Agree and Pay Screen



I allow Pizza Express to automatically charge my PayPal account. If my primary payment method cannot complete a transaction, PalPal will try using other payment methods with my account. To change or cancel, go to your Profile page.

After logging in to PayPal, the buyer enters a code that only the buyer and PayPal know. Later, before you take a preapproved payment, prompt the buyer to enter the preapproval PIN. Then from your web server, include the PIN that the buyer entered with the `Pay` request that you send to PayPal. PayPal recommends that you display the payment reason and payment amount when you prompt the buyer for the preapproval PIN.

3. Submitting Your Application to PayPal

Before you distribute your mobile application publicly, you need an authorized application ID from PayPal. PayPal tests all mobile applications before issuing application IDs. Test your mobile application thoroughly in the PayPal Sandbox by using `APP-80W284485P519543T` as your test application ID. Then, submit your test application to PayPal.

1. Log in or sign up on PayPal's developer website, X.com.

2. After logging in successfully, click the "My Apps" tab.

3. Click **SUBMIT NEW APP**.

4. Fill in the 2-page "Submit New App" form.

   If you need more time, you can save your form as a draft and return later to complete it.

5. Click **Submit**.

Result:

For those using simple or parallel payments, PayPal reviews your application within 24 hours and responds by sending you your `PayPalApplicationID`. Reviewers at PayPal follow up by email with questions, should they arise, before they approve your mobile application. For those using chained payments or preapprovals, the review may take longer.

After completing this task:

Wait until PayPal sends you your application ID. Then, make sure that you update your software with the following changes before you submit your mobile application to Blackberry:

- **Application ID:** in your calls to `initWithAppID`
- **Environment:** in your calls to `initWithAppID`
- **Recipient:** in the `PayPalPayment` object

A. Currencies Supported by PayPal

PayPal uses 3-character ISO-4217 codes for specifying currencies in fields and variables.

Currency	Currency Code
Australian Dollar	`AUD`
Brazilian Real **NOTE:** This currency is supported as a payment currency and a currency balance for in-country PayPal accounts only.	`BRL`
Canadian Dollar	`CAD`
Czech Koruna	`CZK`
Danish Krone	`DKK`
Euro	`EUR`
Hong Kong Dollar	`HKD`
Hungarian Forint	`HUF`
Israeli New Shekel	`ILS`
Japanese Yen	`JPY`
Malaysian Ringgit **NOTE:** This currency is supported as a payment currency and a currency balance for in-country PayPal accounts only.	`MYR`
Mexican Peso	`MXN`
Norwegian Krone	`NOK`
New Zealand Dollar	`NZD`
Philippine Peso	`PHP`
Polish Zloty	`PLN`
Pound Sterling	`GBP`
Singapore Dollar	`SGD`
Swedish Krona	`SEK`
Swiss Franc	`CHF`
Taiwan New Dollar	`TWD`
Thai Baht	`THB`
U.S. Dollar	`USD`

B. Countries and Regions Supported by PayPal

PayPal uses 2-character IS0-3166-1 codes for specifying countries and regions that are supported in fields and variables.

Country or Region	Country or Region Code
AFGHANISTAN	AF
ÅLAND ISLANDS	AX
ALBANIA	AL
ALGERIA	DZ
AMERICAN SAMOA	AS
ANDORRA	AD
ANGOLA	AO
ANGUILLA	AI
ANTARCTICA	AQ
ANTIGUA AND BARBUDA	AG
ARGENTINA	AR
ARMENIA	AM
ARUBA	AW
AUSTRALIA	AU
AUSTRIA	AT
AZERBAIJAN	AZ
BAHAMAS	BS
BAHRAIN	BH
BANGLADESH	BD
BARBADOS	BB
BELARUS	BY
BELGIUM	BE
BELIZE	BZ
BENIN	BJ
BERMUDA	BM
BHUTAN	BT
BOLIVIA	BO
BOSNIA AND HERZEGOVINA	BA
BOTSWANA	BW
BOUVET ISLAND	BV
BRAZIL	BR

Country or Region	Country or Region Code
BRITISH INDIAN OCEAN TERRITORY	IO
BRUNEI DARUSSALAM	BN
BULGARIA	BG
BURKINA FASO	BF
BURUNDI	BI
CAMBODIA	KH
CAMEROON	CM
CANADA	CA
CAPE VERDE	CV
CAYMAN ISLANDS	KY
CENTRAL AFRICAN REPUBLIC	CF
CHAD	TD
CHILE	CL
CHINA	CN
CHRISTMAS ISLAND	CX
COCOS (KEELING) ISLANDS	CC
COLOMBIA	CO
COMOROS	KM
CONGO	CG
CONGO, THE DEMOCRATIC REPUBLIC OF	CD
COOK ISLANDS	CK
COSTA RICA	CR
COTE D'IVOIRE	CI
CROATIA	HR
CUBA	CU
CYPRUS	CY
CZECH REPUBLIC	CZ
DENMARK	DK
DJIBOUTI	DJ
DOMINICA	DM
DOMINICAN REPUBLIC	DO
ECUADOR	EC
EGYPT	EG
EL SALVADOR	SV
EQUATORIAL GUINEA	GQ
ERITREA	ER
ESTONIA	EE
ETHIOPIA	ET
FALKLAND ISLANDS (MALVINAS)	FK

Country or Region	Country or Region Code
FAROE ISLANDS	FO
FIJI	FJ
FINLAND	FI
FRANCE	FR
FRENCH GUIANA	GF
FRENCH POLYNESIA	PF
FRENCH SOUTHERN TERRITORIES	TF
GABON	GA
GAMBIA	GM
GEORGIA	GE
GERMANY	DE
GHANA	GH
GIBRALTAR	GI
GREECE	GR
GREENLAND	GL
GRENADA	GD
GUADELOUPE	GP
GUAM	GU
GUATEMALA	GT
GUERNSEY	GG
GUINEA	GN
GUINEA-BISSAU	GW
GUYANA	GY
HAITI	HT
HEARD ISLAND AND MCDONALD ISLANDS	HM
HOLY SEE (VATICAN CITY STATE)	VA
HONDURAS	HN
HONG KONG	HK
HUNGARY	HU
ICELAND	IS
INDIA	IN
INDONESIA	ID
IRAN, ISLAMIC REPUBLIC OF	IR
IRAQ	IQ
IRELAND	IE
ISLE OF MAN	IM
ISRAEL	IL
ITALY	IT
JAMAICA	JM

Country or Region	Country or Region Code
JAPAN	JP

JERSEY	JE
JORDAN	JO
KAZAKHSTAN	KZ
KENYA	KE
KIRIBATI	KI
KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	KP
KOREA, REPUBLIC OF	KR
KUWAIT	KW
KYRGYZSTAN	KG
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LA
LATVIA	LV
LEBANON	LB
LESOTHO	LS
LIBERIA	LR
LIBYAN ARAB JAMAHIRIYA	LY
LIECHTENSTEIN	LI
LITHUANIA	LT
LUXEMBOURG	LU
MACAO	MO
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF	MK
MADAGASCAR	MG
MALAWI	MW
MALAYSIA	MY
MALDIVES	MV
MALI	ML
MALTA	MT
MARSHALL ISLANDS	MH
MARTINIQUE	MQ
MAURITANIA	MR
MAURITIUS	MU
MAYOTTE	YT
MEXICO	MX
MICRONESIA, FEDERATED STATES OF	FM
MOLDOVA, REPUBLIC OF	MD
MONACO	MC
MONGOLIA	MN
MONTSERRAT	MS
MOROCCO	MA

Country or Region	Country or Region Code
MOZAMBIQUE	MZ
MYANMAR	MM
NAMIBIA	NA
NAURU	NR
NEPAL	NP
NETHERLANDS	NL
NETHERLANDS ANTILLES	AN
NEW CALEDONIA	NC
NEW ZEALAND	NZ
NICARAGUA	NI
NIGER	NE
NIGERIA	NG
NIUE	NU
NORFOLK ISLAND	NF
NORTHERN MARIANA ISLANDS	MP
NORWAY	NO
OMAN	OM
PAKISTAN	PK
PALAU	PW
PALESTINIAN TERRITORY, OCCUPIED	PS
PANAMA	PA
PAPUA NEW GUINEA	PG
PARAGUAY	PY
PERU	PE
PHILIPPINES	PH
PITCAIRN	PN
POLAND	PL
PORTUGAL	PT
PUERTO RICO	PR
QATAR	QA
REUNION	RE
ROMANIA	RO
RUSSIAN FEDERATION	RU
RWANDA	RW
SAINT HELENA	SH
SAINT KITTS AND NEVIS	KN
SAINT LUCIA	LC
SAINT PIERRE AND MIQUELON	PM
SAINT VINCENT AND THE GRENADINES	VC
SAMOA	WS

SAN MARINO	SM
SAO TOME AND PRINCIPE	ST
SAUDI ARABIA	SA
SENEGAL	SN
SERBIA AND MONTENEGRO	CS
SEYCHELLES	SC
SIERRA LEONE	SL
SINGAPORE	SG
SLOVAKIA	SK
SLOVENIA	SI
SOLOMON ISLANDS	SB
SOMALIA	SO
SOUTH AFRICA	ZA
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	GS
SPAIN	ES
SRI LANKA	LK
SUDAN	SD
SURINAME	SR
SVALBARD AND JAN MAYEN	SJ
SWAZILAND	SZ
SWEDEN	SE
SWITZERLAND	CH
SYRIAN ARAB REPUBLIC	SY
TAIWAN, PROVINCE OF CHINA	TW
TAJIKISTAN	TJ
TANZANIA, UNITED REPUBLIC OF	TZ
THAILAND	TH
TIMOR-LESTE	TL
TOGO	TG
TOKELAU	TK
TONGA	TO
TRINIDAD AND TOBAGO	TT
TUNISIA	TN
TURKEY	TR
TURKMENISTAN	TM
TURKS AND CAICOS ISLANDS	TC
TUVALU	TV
UGANDA	UG

Country or Region	Country or Region Code
UKRAINE	`UA`
UNITED ARAB EMIRATES	`AE`
UNITED KINGDOM	`GB`
UNITED STATES	`US`
UNITED STATES MINOR OUTLYING ISLANDS	`UM`
URUGUAY	`UY`
UZBEKISTAN	`UZ`
VANUATU	`VU`
VENEZUELA	`VE`
VIET NAM	`VN`
VIRGIN ISLANDS, BRITISH	`VG`
VIRGIN ISLANDS, U.S.	`VI`
WALLIS AND FUTUNA	`WF`
WESTERN SAHARA	`EH`
YEMEN	`YE`
ZAMBIA	`ZM`
ZIMBABWE	`ZW`